Porting old patch syntax to NCPatcher

This page will help you understand how you can port any patching syntax to NCPatcher's.

You are searching through NSMBHD or NSMB Central and you find a shiny code patch, you rush and put it in the `source` folder of your project only to find that a code patch is not compatible with your code! That sucks.

So what can we do? For this example NSMB E3 Recreation's `PlayerAnims.cpp` code patch will be used.

Step 1 - Investigation

Let's start by taking a look at `PlayerAnims.cpp`.

```
#include <nsmb.hpp>
#include <nsmb/extra/fixedpoint.hpp>

#define NAKED __attribute__((naked))

// Slow down rotation speed
NAKED void repl_02114DFC_ov_0A() { asm("MOV R5, #0xC00\nBX LR"); }

// Walking transition delay
void repl_0211667C_ov_0A() {}

void repl_02116698_ov_0A(Player* player, int id, bool doBlend, Player::FrameMode frameMode,
fx32 speed, u16 frame) {
    // 3.75fx (0x3C00) is the max walk animation speed
    if (speed > (3.75fx / 2)) {
        speed = (3.75fx / 2);
    }

    if (player->animID == 2) {
```

```
    player->setBodyAnimationSpeed(speed);
  } else {
    if (player->animID == 1) {
      fx32 xvel = Math::abs(player->velocity.x);
      if (xvel >= 1.5fx) {
        player->setAnimation(2, doBlend, frameMode, speed, frame);
      } else {
        player->setBodyAnimationSpeed(speed);
      }
    } else {
      player->setAnimation(1, doBlend, frameMode, speed, frame);
    }
  }
}

// Force jump on anim 1
NAKED void nsub_02116A14_ov_0A() { asm("CMP R0, #1\nB 0x02116A18"); }

// Use anim 1
NAKED void repl_02116A2C_ov_0A() { asm("MOV R1, #1\nBX LR"); }
```

We must now wonder, what kind of a patch is this? Is this an `NSMBe` type patch or a `Fireflower` type patch?

By comparing common traits that each patcher uses we can guess what kind of patch type we are dealing with.

`NSMBe` type patches:

- Does not use attributes to declare patches, uses the function name. `void hook_x() {}`
- Patches always follow the format `<PATCH TYPE>_<ADDRESS HEX>_ov_<OVERLAY HEX>` or `<PATCH TYPE>_<ADDRESS HEX>` if you don't need to specify an overlay.
- `PATCH TYPE` can only be `hook`, `repl` or `nsub`.

`Fireflower` type patches:

- Uses attributes to declare patches. `hook(X) void func() {}`
- Patches always follow the format `<PATCH TYPE>(0x<ADDRESS HEX>, 0x<OVERLAY HEX>)` or `<PATCH TYPE>(0x<ADDRESS HEX>)` if you don't need to specify an overlay.
- `PATCH TYPE` can only be `hook`, `rlnk`, `safe` or `over`.

Did you guess correctly what kind of patch we are working with?

Click here to reveal the answer

NSMBe

Step 2 - Porting

This is a fairly simple process. Here is a list that shows the different patch syntax between the patchers:

NSMBe	Fireflower	NCPatcher
hook	safe	ncp_hook
repl	rlnk	ncp_call
nsub	hook	ncp_jump
	over	ncp_over
		ncp_repl

And here is an example comparing some of them:

```
// NSMBe
void hook_02000000() {} // doSomethingPatch
void repl_0200A000() {} // doUnspecifiedPatch
void repl_02010000_ov_0A() {} // doWhateverOverlayPatch
// over does not exist in NSMBe

// Fireflower
safe(0x02000000) void doSomethingPatch() {}
rlnk(0x0200A000) void doUnspecifiedPatch() {}
rlnk(0x02010000, 10) void doWhateverOverlayPatch() {}
over(0x02159348, 52) static int stupidVar = 0x0215CA6C;

// NCPatcher
ncp_hook(0x02000000) void doSomethingPatch() {}
ncp_call(0x0200A000) void doUnspecifiedPatch() {}
ncp_call(0x02010000, 10) void doWhateverOverlayPatch() {}
ncp_over(0x02159348, 52) static int stupidVar = 0x0215CA6C;
```

These addresses are ficticious and purely for demonstration!

An important thing to remember is that all values in `NSMBe` patches are always written in hexadecimal without `0x` prepended to them. In NCPatcher if you want to specify an hexadecimal value you need to prepend `0x`, otherwise the value will be interpreted as a decimal value!

Let's go back to `PlayerAnims.cpp` and try to apply these changes.

```
#include <nsmb.hpp>
#include <nsmb/extra/fixedpoint.hpp>

#define NAKED __attribute__((naked))

NAKED ncp_call(0x02114DFC, 10)
void slowDownRotationSpeed() { asm("MOV R5, #0xC00\nBX LR"); }

// Walking transition delay
ncp_call(0x0211667C, 10) void doNotJumpOnAnim2() {}

ncp_call(0x02116698, 10)
void customPlayerAnimator(Player* player, int id, bool doBlend, Player::FrameMode frameMode,
fx32 speed, u16 frame) {
    // 3.75fx (0x3C00) is the max walk animation speed
    if (speed > (3.75fx / 2)) {
        speed = (3.75fx / 2);
    }

    if (player->animID == 2) {
        player->setBodyAnimationSpeed(speed);
    } else {
        if (player->animID == 1) {
            fx32 xvel = Math::abs(player->velocity.x);
            if (xvel >= 1.5fx) {
                player->setAnimation(2, doBlend, frameMode, speed, frame);
            } else {
                player->setBodyAnimationSpeed(speed);
            }
        } else {
            player->setAnimation(1, doBlend, frameMode, speed, frame);
        }
    }
}
```

```
NAKED ncp_jump(0x02116A14, 10)
void forceJumpOnAnim1() { asm("CMP R0, #1\nB 0x02116A18"); }

NAKED ncp_call(0x02116A2C, 10)
void useAnim1() { asm("MOV R1, #1\nBX LR"); }
```

The code should now compile!

If your code still doesn't work because it complains about some functions not being defined or not existing then you might want to check this out as well: Porting old patches to the NSMB Code Reference

What if the patch was an assembly `.s` file instead of C `.c` or C++ `.cpp`? The process is the same.

```
hook_....:
    BX LR
```

Becomes

```
ncp_hook(...)
    BX LR
```

Step 4 - Tidying up

Even though the code should now be able to execute, it is still not in its optimal state. This part is slightly more complicated because it requires understanding the code.

NCPatcher includes its own definition of `__attribute__((naked))` which is `ncp_asmfunc` so we remove that macro definition and use `ncp_asmfunc` instead.

```
#include <nsmb.hpp>
#include <nsmb/extra/fixedpoint.hpp>

ncp_asmfunc ncp_call(0x02114DFC, 10)
void slowDownRotationSpeed() { asm("MOV R5, #0xC00\nBX LR"); }

// Walking transition delay
ncp_call(0x0211667C, 10) void doNotJumpOnAnim2() {}

ncp_call(0x02116698, 10)
```

```
void customPlayerAnimator(Player* player, int id, bool doBlend, Player::FrameMode frameMode,
fx32 speed, u16 frame) {
    // 3.75fx (0x3C00) is the max walk animation speed
    if (speed > (3.75fx / 2)) {
        speed = (3.75fx / 2);
    }

    if (player->animID == 2) {
        player->setBodyAnimationSpeed(speed);
    } else {
        if (player->animID == 1) {
            fx32 xvel = Math::abs(player->velocity.x);
            if (xvel >= 1.5fx) {
                player->setAnimation(2, doBlend, frameMode, speed, frame);
            } else {
                player->setBodyAnimationSpeed(speed);
            }
        } else {
            player->setAnimation(1, doBlend, frameMode, speed, frame);
        }
    }
}

ncp_asmfunc ncp_jump(0x02116A14, 10)
void forceJumpOnAnim1() { asm("CMP R0, #1\nB 0x02116A18"); }

ncp_asmfunc ncp_call(0x02116A2C, 10)
void useAnim1() { asm("MOV R1, #1\nBX LR"); }
```

Now, take a look at the original purpose of `repl_0211667C_ov_0A` (now named `doNotJumpOnAnim2`) and the code it targeted.

```
ov10:02116678   CMP R0, #2
ov10:0211667C   BEQ 0x021166A0
ov10:02116680   MOV R0, R5
```

We can see that what we are doing is the following:

```
ov10:02116678    CMP R0, #2
ov10:0211667C    BL  repl_0211667C_ov_0A
ov10:02116680    MOV R0, R5
```

```
//...
repl_0211667C_ov_0A:
    BX LR // return generated by the compiler
```

Essentially we are just making it so `BEQ 0x021166A0` will never jump to `0x021166A0`, but we are not doing this efficiently because we jump from `0x0211667C` to `repl_0211667C_ov_0A` and then back to `0x02116680` instead of just continuing. This wastes memory and CPU cycles, but it was the only way of doing so in `NSMBe`. Instead we can write it like `ncp_repl(0x0211667C, 10, "NOP")` in NCPatcher, making the instruction do nothing and just skip to the next one without using any more memory.

```
ov10:02116678    CMP R0, #2
ov10:0211667C    NOP        // Skips to the next instruction
ov10:02116680    MOV R0, R5
```

After evaluating all theses different cases, our optimal code should look like this:

```
#include <nsmb.hpp>
#include <nsmb/extra/fixedpoint.hpp>

// Slow down rotation speed
ncp_repl(0x02114DFC, 10, "MOV R5, #0xC00")

// Walking transition delay
ncp_repl(0x0211667C, 10, "NOP")

ncp_call(0x02116698, 10)
void customPlayerAnimator(Player* player, int id, bool doBlend, Player::FrameMode frameMode,
fx32 speed, u16 frame) {
□// 3.75fx (0x3C00) is the max walk animation speed
□if (speed > (3.75fx / 2)) {
□□speed = (3.75fx / 2);
□}

□if (player->animID == 2) {
□□player->setBodyAnimationSpeed(speed);
□} else {
□□if (player->animID == 1) {
□□□fx32 xvel = Math::abs(player->velocity.x);
□□□if (xvel >= 1.5fx) {
□□□□player->setAnimation(2, doBlend, frameMode, speed, frame);
□□□} else {
□□□□player->setBodyAnimationSpeed(speed);
```

```
□□□}
□□} else {
□□□player->setAnimation(1, doBlend, frameMode, speed, frame);
□□}
□}
}

// Force jump on anim 1
ncp_repl(0x02116A14, 10, "CMP R0, #1")

// Use anim 1
ncp_repl(0x02116A2C, 10, "MOV R1, #1")
```

Revision #4
Created 2024-04-17 07:07:09 UTC by TheGameratorT
Updated 2025-04-14 16:11:02 UTC by Ndymario